UNITED STATES PATENT AND TRADEMARK OFFICE

amv

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/809,499	03/15/2001	Gerard J. Holzmann	Holzmann 17	5667

47394 7590 06/03/2005

HITT GAINES, PC
LUCENT TECHNOLOGIES INC.
PO BOX 832570
RICHARDSON, TX 75083

EXAMINER	
STEELMAN, MARY J	
ART UNIT	**PAPER NUMBER**
2191	

DATE MAILED: 06/03/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)
	09/809,499	HOLZMANN, GERARD J.
	Examiner	Art Unit
	Mary J. Steelman	2191

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *20 December 2004*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-23* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-23* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *20 December 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☒ Other: *Copy of accepted drawing*.

DETAILED ACTION

1. This Office Action is in response to Remarks and Amendments received 20 December 2004. Per Applicant's request, claims 1-4, 6-9, 11-12, 15, 16, and 18-23 have been amended. Claims 1-23 are pending.

Drawings

2. In view of the receipt of Replacement Sheet for Drawing FIF. 1, the prior objection is hereby withdrawn.

Requirement for Information – 37 CFR 1.105

3. Examiner acknowledges the receipt of : Holzmann et al., "Software Model Checking: Extracting Verification Models from Source Code", published in Proceedings of PSTV/FORTE99 (Kluwer, 1999), page 481.

Double Patenting

4. Examiner acknowledges the receipt of a Terminal disclaimer for co-pending Application 09/ 850382 as related to this instant application.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

6. Claims 1- 23 are rejected under 35 U.S.C. 103(a) as being unpatentable over US Patent 6,625,797 to Edwards et al.

Per claim 1:

A method for automatically extracting a verification model from program source code independent of the source code, comprising the steps of:

(Edward discloses a method for extracting a verification model from source code (see for example, Abstract and FIG. 1.) A correct 'target model'/ 'verification model' is extracted from source code.)

-generating a parse tree defining a control flow from the source code for procedural elements thereof;

(Edwards shows extracting information from a source code and defining control flow at col. 3, lines 36-39.)

-identifying source code elements from the parse tree;

(Edwards shows identifying source code elements from the parse tree at col. 5, lines 39-54 which explain that the flow graph (parse tree) is translated via library substitution (substitute source code element) by the core engine using the source code specification or information from the flowgraph through analysis of language semantics and structure.)

-from the parse tree, generating text strings for selected ones of the source code elements that correspond to function statements of the source code;

(Edwards shows, at col. 6, lines 4-14, generating text strings (compiled sequence of bytecodes), that corresponds to function statements of the source code (indicates a specific operation to be performed).)

-defining corresponding default conversions for translating the source strings into a target language of a model checker;

(Edwards disclosed, at col. 5, lines 39-44, translating source strings into target language using default conversions (library substitution…according to the annotations…))

-automatically generating in the target language a verification model that conforms to the control flow and to the corresponding default conversions for the selected ones of the source code elements.

(Edwards disclosed, at col. 5, line 62- col. 6, line 2 & col. 6, lines 26-42, automatically generating a target language verification model (Hardware implementation) from the control flow, using the default conversions (as an example, libraries). Also see Figure 1.)

Edwards failed to explicitly disclose the term 'verification model'. However, he did disclose a (col. 1, line 66-col. 2, line 4) "compilation tool that analyzes the software description to generate a control and data flow graph." Optimizations are preformed in the translation process and correct functionality is ensured (col. 5, lines 22-23). Col. 1, lines 63-64, Edwards invention uses

a high level of abstraction with a semantic model infers the resulting hardware implementation." Thus the model is verified.

Therefore, it would have been obvious, to one of ordinary skill in the art at the time of the invention, to consider that Edward's invention implies a verification model, because he implied that a correct model is developed in the analysis, compilation and translation process.

Per claim 2:

-optionally searching a conversion table for an entry associated with at least one of the text strings, the entry including a translation for the at least one of the text strings;
(Edwards disclosed searching for a text string substitution at col. 9, lines 21-25 & col. 18, lines 11-16, where hardware elements are directly inferred from the source code and mimic source code behavior. It is inherent that a conversion table or analogous data structure is searched to perform a substitution / conversion of code text strings.)

-substituting the translation for the corresponding default conversion for the at least one of the text strings, wherein the verification model further conforms to the translation.
(Edwards discloses default conversion substitutions, as an example, at col. 5, line 41, 'libraries' and col. 6, lines 33-34, 'direct substitution or library references." See Figure 3. Col. 18, lines 15-16, discloses that the structures create a hardware implementation that mimics object behavior found in the original source code (further conforms to the translation))

Per claim 3:

-function statements include elements of the source language selected from the group consisting of: declarations, assignments, function calls, return statements, and Boolean conditions.

(Edwards disclosed, col. 6, lines 8-9, specific operation (function statements)to be performed. Edwards disclosed, as an example Boolean conditions at col. 7, line 29, and function calls / returns at col. 7, lines 40-62 (push / pop).)

Per claim 4:

-generating of text strings includes the further step of expressing the text strings in a canonical form.

(Edwards disclosed, at col. 6, lines 20-25, "...nodes...defined by the specification for the target microprocessor architecture...(canonical form)." The nodes are used to generate target text strings. Grammar rules are used to translate source to target canonical form.)

Per claim 5:

-specifics of the corresponding default conversions can depend on a usage of the selected ones of the source code elements.

(Edwards disclosed corresponding default conversions at col. 6, lines 26-34, which also disclosed that user supplied constraints and preferences (can depend on a usage) are considered.)

Per claim 6:

-conversion table further includes samples of text strings.

(Edwards discloses default conversion substitutions, as an example, at col. 5, line 41, 'libraries' and col. 6, lines 33-34, 'direct substitution or library references." See Figure 3. Edwards disclosed searching for a text string substitution at col. 9, lines 21-25 & col. 18, lines 11-16, where hardware elements are directly inferred from the source code and mimic source code behavior. It is inherent that a conversion table or analogous data structure is searched to perform a substitution / conversion of code text strings. Col. 18, lines 15-16, discloses that the structures (suggestive of conversion tables) create a hardware implementation that mimics object behavior found in the original source code.)

Per claim 7:

-conversion table further includes classes of text strings.

See comments related to a 'conversion table' in rejection of claims 2 & 6 above. Edwards disclosed at col. 6, lines 4-5, suitable code could be Java, C, C++, a type of software that includes 'class' formats. It is inherent that a structure of conversion equivalents would provide "classes of text strings".

Per claim 8:

-searching of the conversion table includes the step of pattern matching the at least one of the text strings to the samples of text strings.

See col. 9, lines 21-25, and col. 18, lines 11-16 where Edwards discloses mappings to match portions of source code (text strings).

Per claim 9:

-searching of the conversion table includes the step of pattern matching the at least one of the text strings to the classes of text strings.

See col. 9, lines 21-25, and col. 18, lines 11-16 where Edwards discloses mappings to match portions of source code (text strings). Mappings between source and 'classes of text strings', directly inferring a hardware implementation, include JAVA, C, and C++ code (col. 6, line 5) (classes of text strings)

Per claim 10:

-corresponding default conversions causes the translating of the text strings to respective equivalent statements in the target language when the selected ones of the source code elements are fully relevant to a property to be tested, the translating of the text strings to null statements in the target language when the selected ones of the source code elements are irrelevant to the property to be tested, and the translating of the text strings to preservation statements in the target language when the selected ones of the source code elements are partially relevant to the property to be tested, preservation statements being statements that preserve a relevant part of the text strings and that suppress an irrelevant part of the source strings.

Edwards disclosed at col. 8, lines 31-39, that corresponding default conversions may be fully relevant (should have a constant value applied) or may preserve a relevant part (substitution of a constant value). Col. 9, lines 9-10 disclose 'translating text strings to null statements…when source code elements are irrelevant…", any logic whose outputs are never used is removed."

Per claim 11:

-generating a verification model step includes the further step of translating ones of the text strings to a non-deterministic choice of possible outcomes.

As an example, Edwards disclosed that a non deterministic choice may be available at col. 12, line 61- col. 13, line 12, where control flow branches suggest multiple possible choices of outcomes.

Per claim 12:

-generating a verification model step includes the step of populating the control flow with the translated text strings.

Edwards, see FIG. 1 where control flow graph is generated (verification model / populating with translated text strings). Edwards, col. 6, lines 11-13, "…each bytecode (control flow populated with translated text strings) is now defined to cause the instantiation of a specific hardware circuit in the final hardware implementation…"

Per claim 13:

-default conversion includes a keep, the keep causing the generating of a verification model step to provide an equivalent statement in the target language.

Edwards disclosed at col. 18, lines 14-16, that a 'keep' default conversion, causes the generating of a verification model providing an equivalent statement in the target language (hardware mimics original source code)

Per claim 14:

-default conversion comprises a hide, the hide causing the generating of a verification model step to provide a null statement in the target language.

Edwards does disclose (col. 8, lines 3-5) that a target node that does not consume data will have the control signal removed (a hide / a null statement). And at col. 8, lines 46-50, certain optimization techniques result in removal (hide) of code when translating from the source (provide a null statement in the target language.)

Per claim 15:

-default conversion comprises a print, the print causing the generating of a verification model step to embed the respective text strings in a print statement in the target language.

Edwards suggested (col. 3, lines 23-24) that nodes may be annotated with supplementary information concerning particular conditions to be met when generating a hardware implementation (embed a text string in target language). Edwards suggested (col. 5, lines 39-44) that an annotated logic design flow graph is translated into the target language via library substitution (embed text strings in print statement in target language), circuit generators, inline code (embed text strings in print statement in target language), library references, etc., according to the annotations of each node and path.

Per claim 16:

-simplifying parse tree according to translated text strings.

Edwards disclosed simplifying the parse tree at col. 8, line 40 – col.10, line 15, whereby various optimizations used in translating text strings are described.

Per claim 17:

-simplifying step includes the steps of: removing nodes corresponding to null statements; removing nodes successive to false nodes; skipping selected nodes mapped to true.

Edwards does disclose (col. 8, lines 3-5) that a target node that does not consume data will have the control signal removed (remove nodes successive to false nodes). And at col. 8, lines 46-50, certain optimization techniques result in removal of code when translating from the source (provide a null statement in the target language), including removing redundant logic (skipping selected nodes mapped to true).

Per claim 18:

-collecting certain data object information for nodes in the parse tree corresponding to function statements in the source code, the certain data object information including definition information and use information;

Edwards disclosed collecting data object information at col. 2, lines 30-51, whereby the compiled version of source code is analyzed to generate a functional hardware implementation by creating a control flow graph. The relationships between operations (corresponding to

function statements in source code) are detailed. Information describing operations required, the order and dependencies is represented (collecting certain data object information for nodes).

-constructing a data dependency graph for the source code based upon the collected data object information, the data dependency graph having data dependency graph nodes corresponding to a data object, the data dependency graph having directed edges from first data dependency graph nodes to successive data dependency graph nodes if the successive data dependency graph nodes are used at least, once in a definition of the first data dependency graph nodes;
Edwards disclosed a data dependency graph at col. 2, lines 49-51 , "Control and Data Flow Graph". Nodes represent operations and paths (directed edges) represent logical directed relationships between nodes. Col. 2, lines 62-65, "The second type of path in a flowgraph is a data path. These paths indicate the relationships between data producers and data consumers...(data dependency graph)" Edwards disclosed at col. 9, lines 9-10, that data logic whose outputs are never used are removed, thus implying for remaining nodes that" successive data dependency graph nodes are used at least, once in a definition of the first data dependency graph nodes."

-determining a transitive closure for the data dependency graph dependency relation;
Edwards disclosed optimization techniques at col. 8, line 40-through col. 10, line 15, whereby a complete control and data flow graph is produced (transitive closure for the data dependency graph dependency relation).

-adding edges to the data dependency graph according to the transitive closure, the adding step providing a second data dependency graph;

Edwards disclosed edges at col. 2, line 51 (logical directed relationships between nodes). As an example, Adding edges are disclosed at col. 4, lines 64-67, which details a control path looping back upon itself. Another example would be (col. 7, line 67) a conditional node with multiple control outputs.

-for nodes corresponding to the function statements in the source code having translations other than hide or print, marking second data dependency graph data objects with identifiers corresponding to the definition information and the use information;

Edwards disclosed annotating nodes (marking data dependency graph data objects with identifiers) (col. 6, lines 26-42) including user supplied constraints and preferences. Edwards recognized the importance of 'dependencies' between nodes (col. 2, lines 39-45) when producing a Control and Data Flow Graph (col. 2, line 49).

-for nodes corresponding to the function statements in the source code having a hide translation; marking second data dependency graph data objects with a hide identifier; checking the second data dependency graph, data objects for identifiers and the hide identifier.

Edwards disclosed annotating nodes (marking data dependency graph data objects with identifiers) (col. 6, lines 26-42) including user supplied constraints and preferences. Edwards recognized the importance of 'dependencies' between nodes (col. 2, lines 39-45) when producing

a Control and Data Flow Graph (col. 2, line 49) (data dependency graph). Edwards does disclose (col. 8, lines 3-5) that a target node that does not consume data will have the control signal removed (hide / a null statement). And at col. 8, lines 46-50, certain optimization techniques result in removal (hiding) of code when translating from the source (provide a null statement in the target language. A 'second data dependency graph' may refer to additional branched code related to 'non-deterministic' conditionals (col. 8, line 1), or loops that branch back on themselves (col. 4, lines 64-67).

Per claim 19:

-A method for verifying that a software based system satisfies certain properties...

See Edwards Abstract and FIG. 1 which show and extraction model for translating software.

Col. 2, line 27, "...method..."

-automatically extracting a finite state model from the source code, the extracting step including the steps for:

Edwards disclosed automatically extracting a finite state model form source code at col. 1, lines 58-67. The Control and Data Flow graph (col. 2, line 49) which reflects states as data and control, are propagated (col. 3, lines 54-56), loops maintain state (col. 4, lines 65-67).

-abstracting the source code statements based upon relevancies between the certain properties and the source code statements;

Edwards disclosed 'abstracting the source codes statements' by constructing a Control and Data Flow graph (shows relevancies between certain properties and source code statements). Col. 2, lines 34-36, the process involved the creation of an intermediate representation (abstracted source code statements).

-expressing the finite state model in an input language for a logic model;
See Edwards Figure 1, "Control and Data Flow Graph', which depicts a model of behavior on a state. States are disclosed at col. 3, lines 54-56 and col. 4, lines 65-67. Input language for logic model is disclosed at col. 6, lines 4-8.

-checking the finite state model for the certain properties in the logic model checker.
Edwards disclosed (Col. 5, lines 22-24) that elements are inserted to ensure correct (checking certain properties) functionality is preserved in the model of behavior of the states (finite state model). At col. 3, lines 22-24, Edwards disclosed that nodes may be annotated with supplementary information concerning particular conditions to be met (checking certain properties in logic model checker) when generating a hardware implementation.

Edwards failed to explicitly disclose the term 'finite state model'. However, he did disclose a (col. 1, line 66-col. 2, line 4) "compilation tool that analyzes the software description to generate a control and data flow graph", with considerations given to various 'states' in the control flow (col. 3, lines 54-56 and col. 4, lines 65-67). Optimizations are preformed in the translation process and correct functionality is ensured (col. 5, lines 22-23) at the various states. Col. 1,

lines 63-64, Edwards invention uses a high level of abstraction with a semantic model that infers the resulting hardware implementation. Thus the model is verified, preserving state.

Therefore, it would have been obvious, to one of ordinary skill in the art at the time of the invention, to consider that Edward's invention implies a 'finite state model', because he implied that a correct model, including correct states, is developed in the analysis, compilation and translation process.

Per clam 20:

-A verification system for verifying that a source code system satisfies certain properties, the verification system comprising:

Edwards: col. 26, lines 27-48, "…apparatus (system)…"

-a model extractor operable to automatically extract a finite state model from the source code, the model extractor implementing default conversions for translating selected source code elements and including:

Edwards disclosed, at col. 2, lines 17-51, that source code input is extracted to a model translated source code statements, using a Control and Data Flow Graph (CDFG). The CDFG represents states (col. 3, lines 54-56 and col. 4, lines 65-67 / finite state model). Default conversions are disclosed at col. 5, lines 39-44, "…library substitution…"

-a table of translations for translating other selected source code elements based upon defined abstractions;

Edwards disclosed, as an example, col. 5, lines 39-44, "The fully resolved, elaborated and annotated logic design flow graph representation is translated into the target language via library substitution, circuit generators, inline code, library references (other source code elements), etc., according to the annotations of each node and path." It is inherent that 'library substitutions' and 'library references' imply that a type of structure (table) is searched to find the correct translation for source code elements.

-a translator responsive to the translations of the selected source code elements and the other selected source code elements for expressing the finite state model in an input language for a logic model checker;

Edwards disclosed at col. 6, lines 4-9, "...algorithms are coded in a high-level source code (Java, C, C++, Pascal, etc.) and compiled to a sequence of bytecodes which are to be executed...Each bytecode...indicates a specific operation to be performed...each bytecode is now defined to cause the instantiation of a specific hardware circuit in the final hardware implementation...through a two step compilation-translation (translator responsive to the translations of the selected source code elements) process. The logical model checker is captured in the Control and Data Flow Graph (CDFG) (col. 2, line 49) which ensures functionality (col. 5, lines 22-23).

-a logic model checker responsive to the certain properties and the finite state model for checking the finite state model for the certain properties.

As an example, Edwards disclosed a CDFG (col. 2, line 49) (finite state model) representing a translation from a source input, ensuring correct functionality (col. 5, lines 22-23), using substitutions for text strings (col. 5, lines 39-44) and giving consideration to user specified annotations (col. 3, lines 22-35) that provide supplementary information concerning particular conditions. Edwards disclosed (col. 1, lines 66-67) "a compilation tool (logic model checker) that analyzes the software description to generate a control and data flow graph."

Edwards failed to explicitly disclose the terms 'finite state model' and 'logic model checker'. However, he did disclose a (col. 1, line 66-col. 2, line 4) "compilation tool (logic model checker) that analyzes the software description to generate a control and data flow graph", with considerations given to various 'states' in the control flow (col. 3, lines 54-56 and col. 4, lines 65-67). Optimizations are preformed in the translation process and correct functionality is ensured (col. 5, lines 22-23) at the various states. Col. 1, lines 63-64, Edwards invention uses a high level of abstraction with a semantic model that infers the resulting hardware implementation. Thus, using the tool (logic model checker) the model is verified, preserving state.

Therefore, it would have been obvious, to one of ordinary skill in the art at the time of the invention, to consider that Edward's invention implies a 'finite state model' and 'logic model

checker', because he implied that a correct model, including correct states, is developed in the analysis, compilation and translation process, using a tool.

Per claim 21:

-the model extractor further includes a parser for constructing a parse tree from the source code;

Edwards disclosed parsing at col. 4, lines 40-45. Parsed code is used to populate the CDFG. Col. 5, lines 39-40, "The fully resolved, elaborated and annotated logic design flow graph (parse tree) is translated..."

-wherein the translator translates selected text strings generated form the parse tree.

Edwards disclosed (Col. 5, lines 30-44) that source strings are translated to target language via library substation, circuit generators, inline code, library references, etc., according to the annotations of each node and path.

Per claim 22:

-the model extractor further operates to provide a control flow from the parse tree and to populate the control flow with translated text strings.

Edwards disclosed, col. 1, lines 66-67, "a compilation tool (model extractor) that analyzes the software description to generate a control and data flow graph (provide a control flow from the parse tree). This graph is then the intermediate format used for optimizations, transformations and annotations. The resulting graph is then translated (populate with translated text strings)..."

Per claim 23:

-A method for automatically extracting a verification model from source code having a control flow for procedural elements of the source code, the method independent of the source code and comprising the steps of:

Edwards, col. 2, line 27, "…method…" Edwards disclosed a tool for extracting a verification model (CDFG) from source code (col. 1, line 66 – col. 2, line 4 & col. 2, line 49).

-generating selected text strings from the source code that represent the control flow;

Edwards disclosed generating selected text strings from the source code at col. 1, line 66 – col. 2, line 4, "analyzing…bytecodes in the compiled source, and the action of each node with respect to control flow (representing control flow)…"

-translating ones of the selected text strings to corresponding target language statements according to default conversions;

Edwards disclosed translating strings corresponding to default conversions at col. 2, lines 33-34, "…direct substitution (default conversion)…"

-optionally searching a conversion table for user defined entries associated with the selected source strings, the conversion table including a plurality of translations associated with various ones of the text strings;

Edwards suggested that user defined flowgraph annotations (col. 6, lines 30-32) may be used in the translations. More details on the annotations may be found at col. 3, lines 21-35. It is inherent that tables may represent a structure used to search for stored user supplied annotations, when converting source strings.

-translating other ones of the selected source strings to corresponding target language statements according to the user defined entries;

Edwards, col. 6, lines 26-42 explain the substitution of text strings corresponding to target language statements. User defined constraints and preferences are supplied as annotations to the nodes (col. 3, lines 21-35).

-automatically populating the control flow with the target language statements.

Edwards disclosed a tool (automatically) to analyze, and translate a control flow with target statements (col. 1, line 66-col. 2, line 4). Col. 6, lines 26-42 explain the substitution (populating) with target language statements.

Edwards failed to explicitly disclose the term 'verification model'. However, he did disclose a (col. 1, line 66-col. 2, line 4) "compilation tool (logic model checker) that analyzes the software description to generate a control and data flow graph" (verification model). Optimizations are preformed in the translation process and correct functionality is ensured (col. 5, lines 22-23). Col. 1, lines 63-64, Edwards invention uses a high level of abstraction with a semantic model

that infers (verification model) the resulting hardware implementation. Thus, using the tool the model is verified.

Therefore, it would have been obvious, to one of ordinary skill in the art at the time of the invention, to consider that Edward's invention implies a 'verification model', because he implied that a correct model, is developed in the analysis, compilation and translation process, using a tool.

Response to Arguments

7. Applicant's arguments with respect to claims 1-23 have been considered but are moot in view of the new ground(s) of rejection.

Conclusion

8. Applicant's amendment necessitated the new grounds of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event,

however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

9. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Mary Steelman, whose telephone number is (571) 272-3704. The examiner can normally be reached Monday through Thursday, from 7:00 AM to 5:30 PM If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached at (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Mary Steelman *Mary Steelman*

05/18/2005

WEI Y. ZHEN
PRIMARY EXAMINER

Accepted by [signature] 5/17/05 MS

FIG. 1